My Adventures with Simulation in Matlab (Illustrated Edition)

March 14, 2015

1 Abstract

2 Declaimer

This article might contain a fair amount of grammatical errors, misused words and other blatant mistakes. Please forgive me :(.

3 Goals

Describing a different and non-analytical approach to our problem and improve my computer graphic abilities.

4 Introduction

A while ago it occurred to me that since we (hopefully) know all the equations which describe the system in any given time, we can simply feed it into a computer and let it do the hard work. In other words, I thought that we simply can run our biological system using computer simulation in matlab.

I used this guideline when I wrote my simulation last Wednesday. I tried to implement our system in matlab, using the rules we had formulated together, AKA 'the monstrous ode system'.

On the theoretical part of this assay I will explain every principle twice: first using a simplified system with just one variable, and second showing the general case. Statements regarding the simple case will be denoted with an asterisk.

Part I
Theory

5 Problem Definition

We succeeded (more or less) in describing our system using mathematical equations (more about this issue later). The mathematical model we developed is consisted of initial condition (the state of the system in t=0) and a bunch of equations that tells us the derivatives with respect to time of the variables (which are, in fact, the amounts of the materials), given the variables at a certain moment. Formally speaking,

Proposition 1. ** The Simplified Problem:*

m is a variable that interests us, and is changing over time.
we know that $m(0) = m_0$, and that $\dot{m} =$
$f(m)$. we are looking for the function $m(t)$.

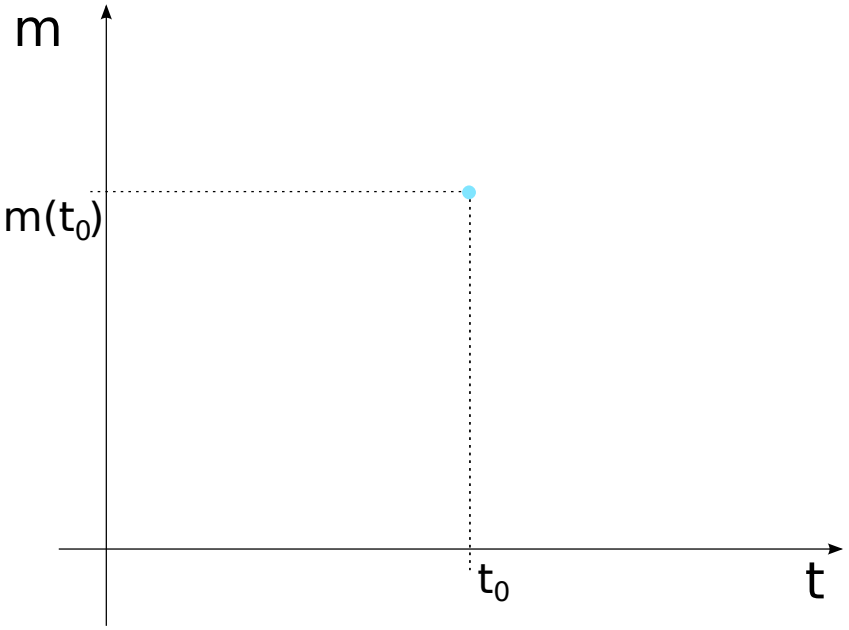
Proposition 2. *The Problem:*

$\overrightarrow{M}$ is a vector of variables that interest us, and is changing over time.
We know that $\overrightarrow{M}(0) = \overrightarrow{M}_0$,and that $\dot{\overrightarrow{M}} =$
$\overrightarrow{F}(\overrightarrow{M})$. we are looking for the function $\overrightarrow{M}(t)$.

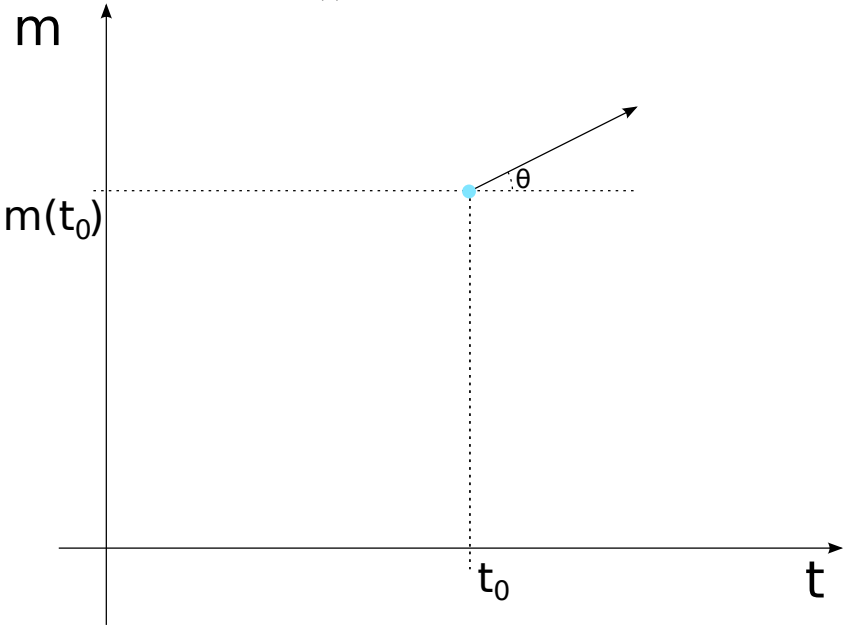
6 Algorithm

Here we describe algorithm to get accurate estimation of a solution to the problems that have been described in the previous section.

First, We shall look at the simplified problem, described in proposition 1. We are looking for the graph of m(t). Assume that we know m's value in a certain moment t_0, namely $m(t_0)$. We shall explain how to compute an approximation to m(t), which will be denoted by $\tilde{m}(t)$.



we know the derivative at this point: $\mathring{m}(t_0) = f(m(t_0))$, therefore we know the tangent to the graph of m(t) in this point.



Where

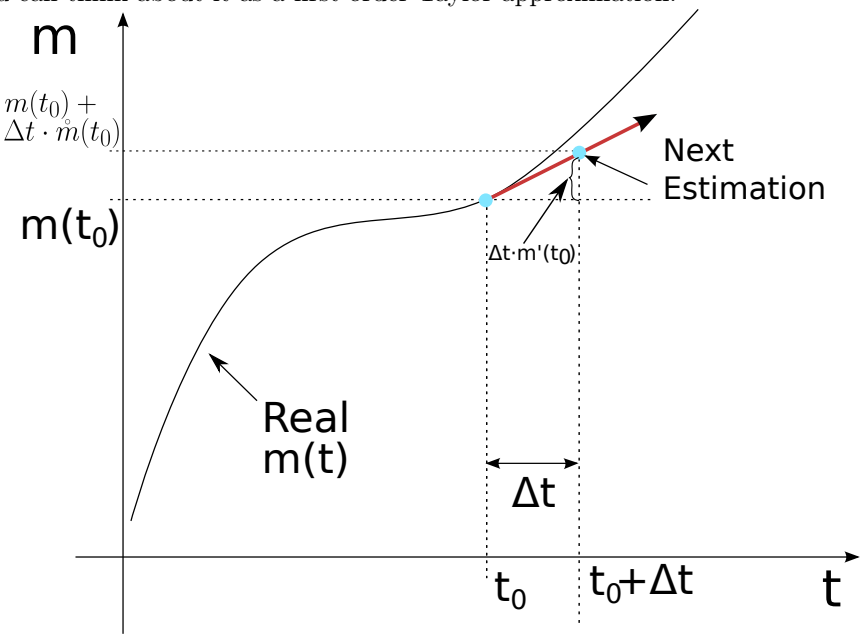
$$\tan\theta = f(m(t_0)) \tag{1}$$

We now want to know the value of the solution a little while after t_0, say

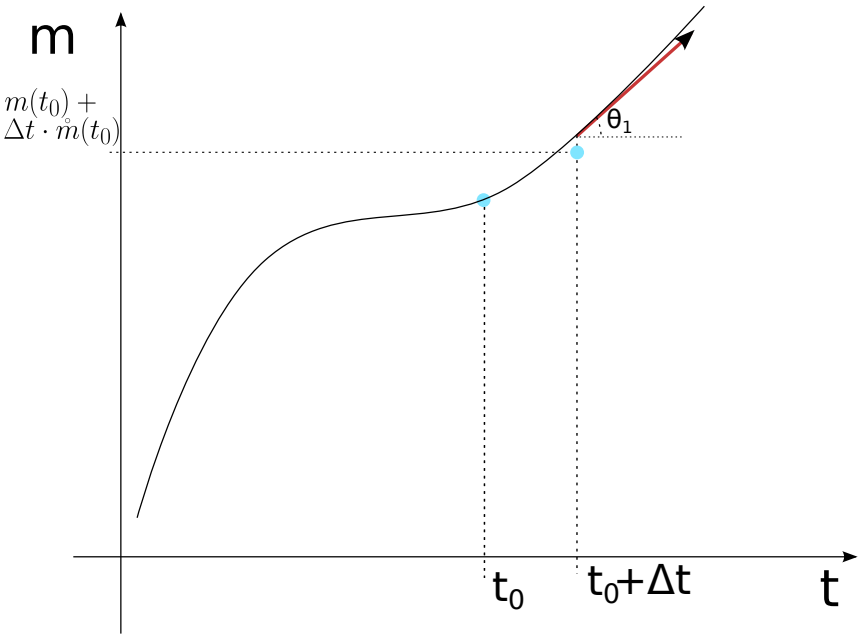
in $t_0 + \Delta t$. The best guess we can take, based on what we know, is to take $m(t_0 + \Delta t)$ to be

$$m(t + \Delta t) \simeq m(t_0) + \Delta t \cdot \mathring{m}(t_0) = m(t_0) + \Delta t \cdot f(m(t_0)) \triangleq \tilde{m}(t + \Delta t) \qquad (2)$$

You can think about it as a first order Taylor approximation.



We now want to estimate the solution a little while later, in $t_0 + 2\Delta t$. We repeat the process that has been described above: We know the tangent to m(t)'s graph at $t_0 + \Delta t$:



with

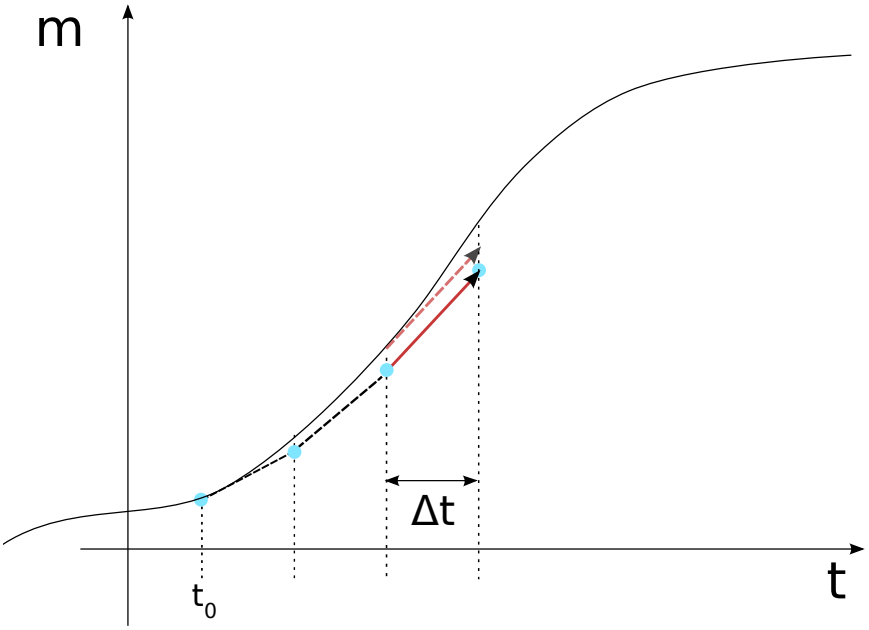
$$\tan\theta_1 = f(m(t_0 + \Delta t)). \tag{3}$$

Note that we don't know $m(t_0 + \Delta t)$, but we can use instead the approximation we have found:

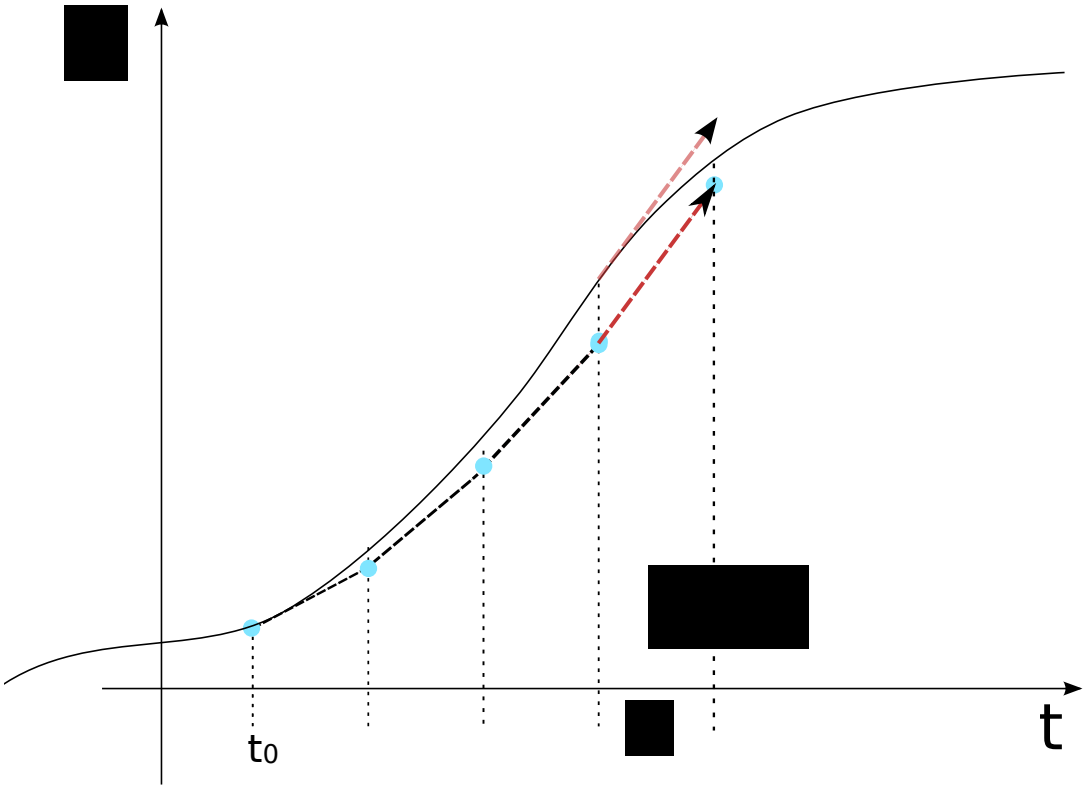
$$\tan\theta_1 \simeq f(\tilde{m}(t + \Delta t)) = f(m(t_0) + \Delta t \cdot \mathring{m}(t_0)). \tag{4}$$

With this approximation, we can get an approximation to the value of $m(t_0 + 2\Delta t)$, which is

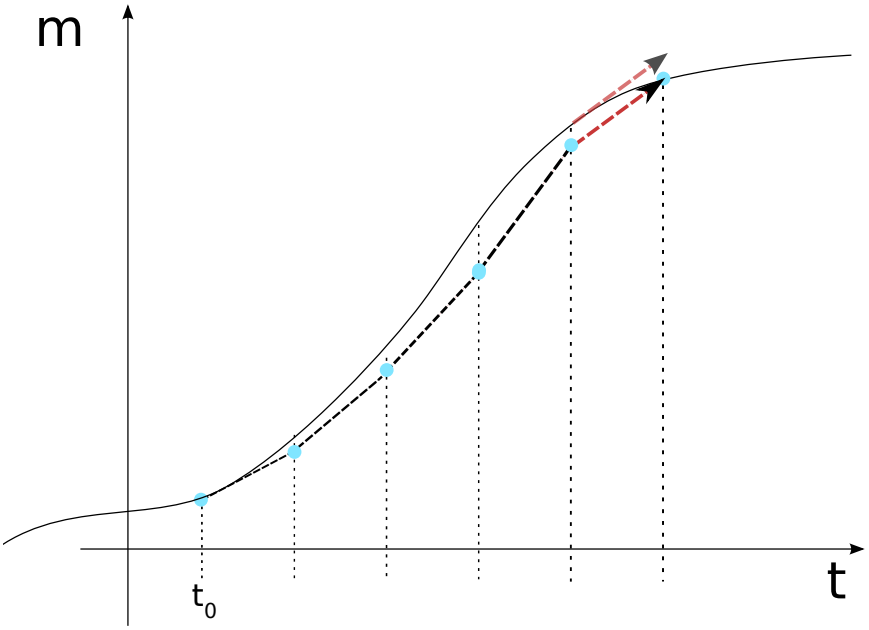
$$m(t_0{+}2\Delta t) \simeq m(t_0{+}\Delta t){+}\Delta t{\cdot}\mathring{m}(t_0{+}\Delta t) \simeq \tilde{m}(t{+}\Delta t){+}\Delta t{\cdot}f(\tilde{m}(t{+}\Delta t)) \triangleq \tilde{m}(t{+}2\Delta t) \tag{5}$$



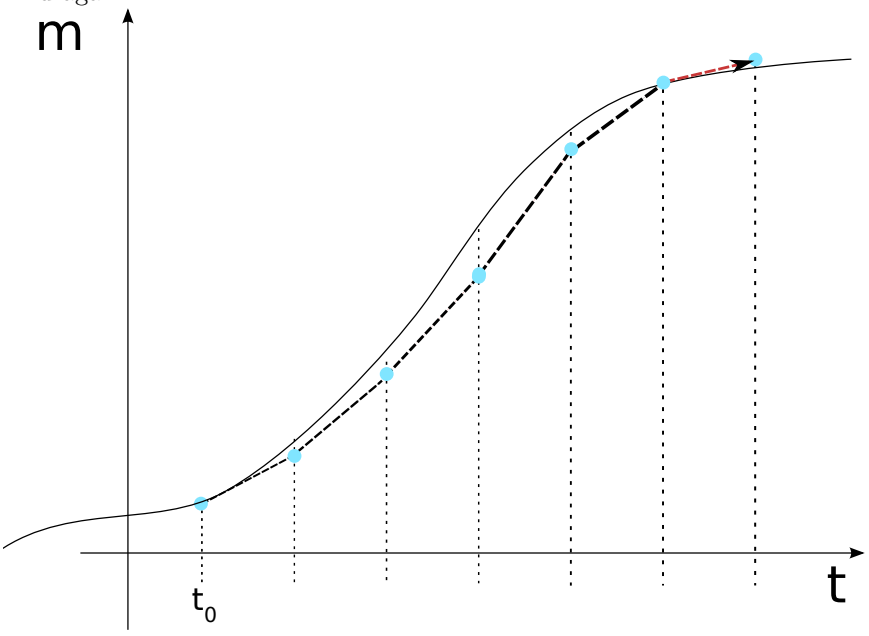
We can repeat it again,



And again,



And again.



I hope that this idea is clear.
This yields the following algorithm:
~

Algorithm 1 Approximate Solution to the simplified problem

inputs: integer n (number of iterations=algorithm repetitions).
float Δt
float m_0(the value of m(t) at t=0)
equation $\mathring{m} = f(m)$

1: $m(0) = m_0$
2: **for** i from 1 to n **do**
3: $m(i \cdot \Delta t) = m((i-1) \cdot \Delta t) + \Delta t \cdot f(m((i-1) \cdot \Delta t)$
4: **end for**
5: m(t) for $t \in \{0, \Delta t, 2\Delta t, ...n\Delta t\}$

Now, we have to prove that when $\Delta t \to 0$ the result of the algorithm $m(t)$ converges to a solution of the problem over the time interval $[0, \tau]$. This is left as an exercise to the reader.

Just kidding, *proof*: denote $\Delta m(t) = m(t) - \tilde{m}(t)$ (We know that the exact solution m(t) exists by Existence and Uniqueness Theorem). Let's assume that we know $\Delta m(k \cdot \Delta t)$ for a certain k. We also know that:

$$\begin{aligned} m(k \cdot \Delta t + \Delta t) &= m(k \cdot \Delta t) + \Delta t \cdot \mathring{m}(k \cdot \Delta t) + o((\Delta t)^2) = \\ &= \tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t) + \Delta t \cdot f(m(k \cdot \Delta t)) + o((\Delta t)^2) = \\ &= \tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t) + \Delta t \cdot f(\tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t)) \\ &\quad + o((\Delta t)^2) \end{aligned}$$

so,

Important Note: The following mathematical derivation does not have the slightest significance to our project. If you don't enjoy long chains of equation, skip it.

$$\begin{aligned} &|m(k \cdot \Delta t + \Delta t) - \tilde{m}(k \cdot \Delta t + \Delta t)| = \\ &= |\tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t) + \Delta t \cdot f(\tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t)) - (\tilde{m}(k \cdot \Delta t) + \Delta t \cdot f(\tilde{m}(k \cdot \Delta t))) + o((\Delta t)^2)| \\ &= |\Delta m(k \cdot \Delta t) + \Delta t \cdot f(\tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t)) - \Delta t \cdot f(\tilde{m}(k \cdot \Delta t)) + o((\Delta t)^2)| \\ &= |\Delta m(k \cdot \Delta t) + \Delta t \cdot (f(\tilde{m}(k \cdot \Delta t) + \Delta m(k \cdot \Delta t)) - f(\tilde{m}(k \cdot \Delta t))) + o\left((\Delta t)^2\right)| \\ &= |\Delta m(k \cdot \Delta t) + \Delta t \cdot \left(f(\tilde{m}(k \cdot \Delta t)) + \Delta m(k \cdot \Delta t) f'(\tilde{m}(k \cdot \Delta t)) + o\left((\Delta m(k \cdot \Delta t))^2\right) - f(\tilde{m}(k \cdot \Delta t))\right) \\ &\qquad + o((\Delta t)^2)| \\ &= |\Delta m(k \cdot \Delta t) + \Delta t \cdot \left(\Delta m(k \cdot \Delta t) f'(\tilde{m}(k \cdot \Delta t)) + o\left((\Delta m(k \cdot \Delta t))^2\right)\right) + o((\Delta t)^2)| \\ &= |\Delta m(k \cdot \Delta t)| \cdot |1 + \Delta t \cdot f'(\tilde{m}(k \cdot \Delta t)) + \frac{\Delta t \cdot o\left((\Delta m(k \cdot \Delta t))^2\right) + o((\Delta t)^2)}{\Delta m(k \cdot \Delta t)}| \\ &\simeq |\Delta m(k \cdot \Delta t)| \cdot |1 + \Delta t \cdot f'(\tilde{m}(k \cdot \Delta t))| \end{aligned}$$

We denote $S = \max_k(|1 + \Delta t \cdot f'(\tilde{m}(k \cdot \Delta t))|)$ and then $|\Delta m((k+1) \cdot \Delta t)| \leq S \cdot |\Delta m(k \cdot \Delta t)|$, so $|\Delta m(k \cdot \Delta t)| \leq S^k |\Delta m(\Delta t)| = S^k |\Delta t \cdot f(0)| \leq S^n |\Delta t \cdot f(0)|$.

When $\Delta t \to 0$ then n increases, but S is still bounded (I'm too lazy to prove it, but it follows somehow from the fact that f is continuous and bounded over $[0, \tau]$. Do it yourself!). We get that the limit: $\lim_{\Delta t \to 0} \Delta m(k \cdot \Delta t)$ is bounded by $\lim_{x \to 0}(1+x)^{1/x} x = e \cdot 0 = 0$, so we kind of proved that the error tends to zero when $\Delta t \to 0$.

End of mathematical derivation.

The general algorithm is derived analogously.

Algorithm 2 Approximate Solution to the problem

inputs: integer n (number of iterations=algorithm repetitions).
float Δt
vector of floats $\overrightarrow{m}_0$(the value of m(t) at t=0)
equation $\overrightarrow{\dot{m}} = \overrightarrow{F}(\overrightarrow{m})$

1: $\overrightarrow{m}(0) = \overrightarrow{m}_0$
2: **for** i from 1 to n **do**
3: $\quad \overrightarrow{m}(i \cdot \Delta t) = \overrightarrow{m}((i-1) \cdot \Delta t) + \Delta t \cdot \overrightarrow{F}(\overrightarrow{m}((i-1) \cdot \Delta t)$
4: **end for**
5: $\overrightarrow{m}(t)$ for $t \in \{0, \Delta t, 2\Delta t, ...n\Delta t\}$

Proving *this* is left as an exercise.

Part II
Application

7 The Matlab Code

I wrote a code in matlab that implements the aforementioned algorithm.

The program performes exactly the steps described in Algorithm 2, with respect to the problem that we have described with our equations, and then draw a graph of the results.

The code is waiting for your review in the Google drive shared folder.

You can take a look at the code and run it yourself, but I strongly advice you against it because I think that the code is a nice showcase of bad coding habits. for example, I put all our variables in one vector together M (stands for materials) which makes our first equation:

$$\frac{dA_{out}}{dt} = -c_1 a a_{in} A_{out} - c_2 A_{out} + c_4 L A_{out} - c_3 L_{in} \cdot A_{out}$$

look like this

```
D(1)=-C(1)*M(2)*M(1) - C(2)*M(1)+C(4)*M(6)-C(3)*M(4)*M(1).
```

Anyway, I'll give here a superficial documentation of my program. The program is consisted of four main functions:

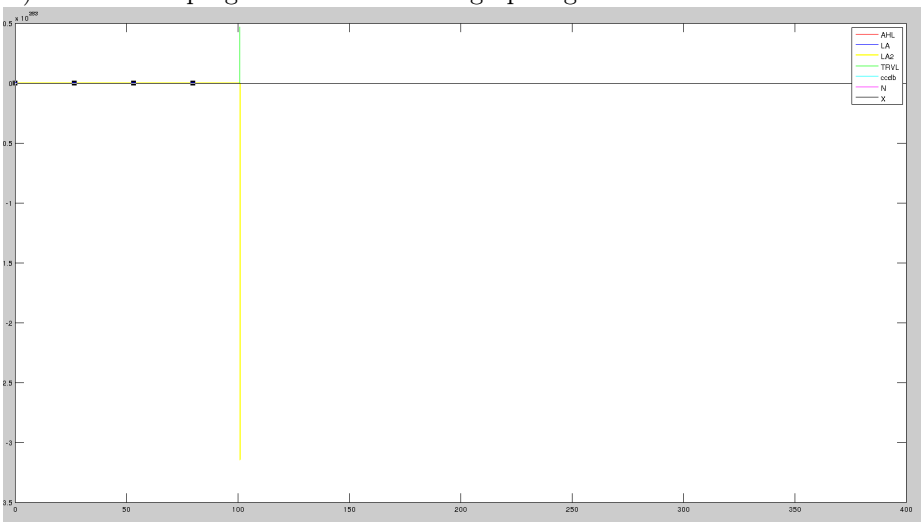
- setDefConst - sets the default values to all the constants. These values are hard-coded in the file.
- derive - returns the derivative of the variables, based on the current state and our equations.
- iterate - updates the variables according to the computed derivatives.
- graph - graphs the results over time.

And the script 'main' is a test programme. M is 1X15 vector of the materials, C is 1X17 of the c-ish constants, V=[v1,v2], U=[u1,u2], N_{max}, and A and B stand for A_{rbs} and B_{rbs} respectively. If you want to use the programme, you can either use the defaults and run the programme 'main' or comment the line that calls to setDefConst and set the constants C,V,U,N_{max},A and B manually according to the documentation in the code, and then run 'main'. But anyway, you would better ask me to do it on Monday.

In order to run the programme, we need (at least right now) to guess the values of the constants. My and Ido's guesses are in "/Modeling Rullz/Constants.xlsx" in the drive.

8 Results

Cheerfully, I guessed some constants (columns 'Itay's Guess 1&2' in the spreadsheet) and ran the programme. This is the graph I got as result:



Something is clearly wrong.

9 Improvements

After I got this nonsensical result I started to search for possible mistakes and repair them.

A summary of my findings:

- What stroke me as a serious problem is the fact that LA2 gets immensely big, which contradicts a fundamental physical law:

Fact 1. *(The law of conservation of AHL) In a closed system, the amount of AHL can't increase.*

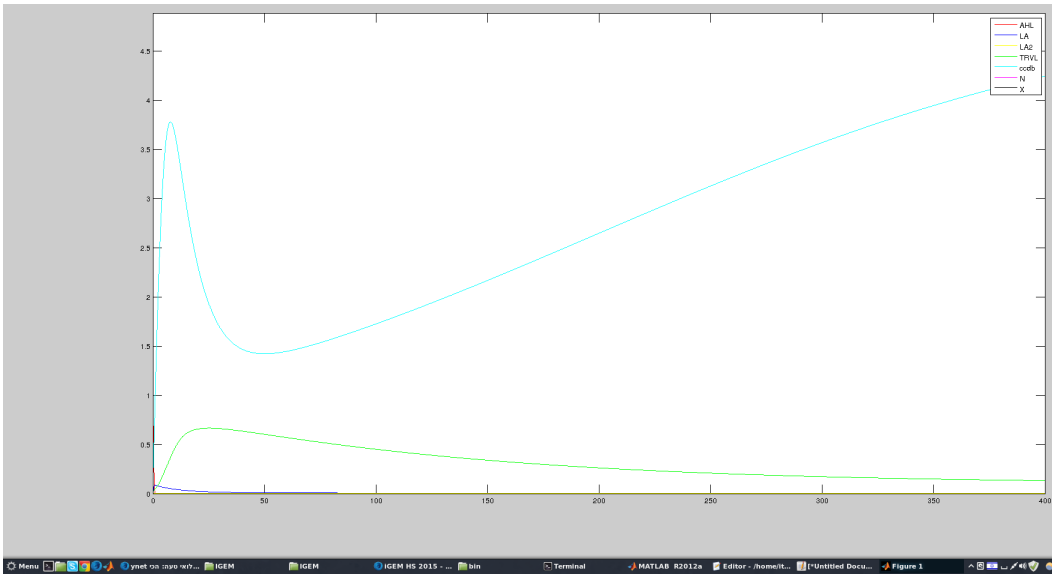
So my guess was that the equations are wrong. I Checked our equations and found 4 things that we didn't take into acount. The updated equations are here: `https://www.overleaf.com/read/ggrvbfsrrypj`.

- I considered the option that my guesses are wrong enough to make the system fail in the simulation. I asked Ido to guess the constants (columns 'Ido's Guess 1' in the spreadsheet). I continued to improve his guesses, until I got a guess that worked.

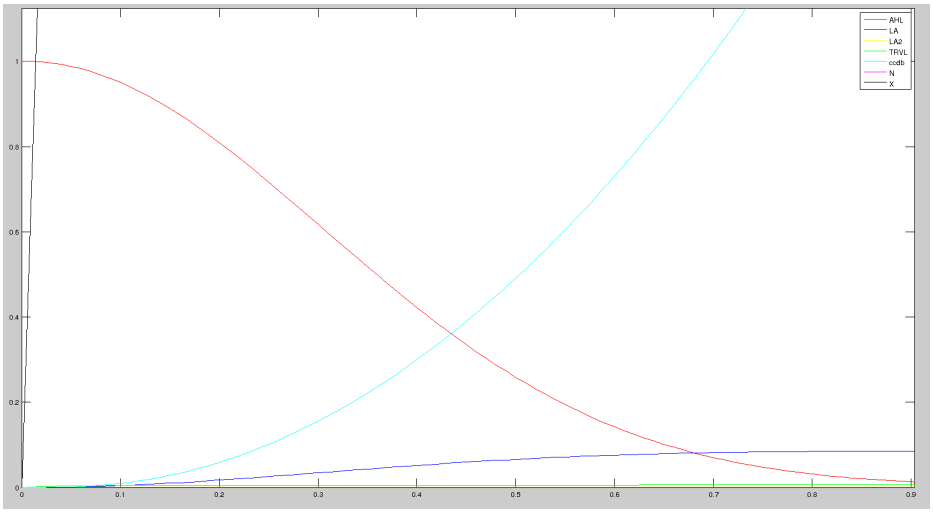
After this, I got some better results.

10 Results #2

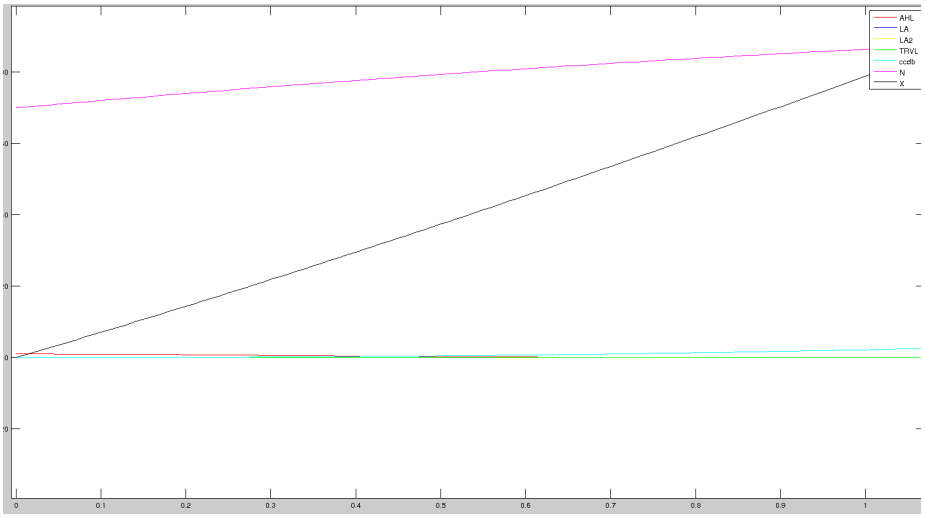
These are some graphs that I got with my program.



Example 1.

Example 2. 

c
d

Example 3. 

More images are here.

11 Discussion

Let's take a look at graphs 1-3. Graph 3 is typical for the zoomed-out graphs I've got, and we can see that over time, the enzyme X increases more or less linearly and N(t) is as it should be. Graph 2 and Graph 3 originated from different parameters. In graph 2 we see that the ccdb increases right from the beginning, so the bacteria die right on the start, hence the system will fail. Graph 1 is the

good type: the ccdb has a peak on the beginning, thenit decreases, and when the TRLV starts to run out it increases again.

Unfortunately, this algorithm is very sensitive to change of the parameters, and for some values of A_0 and too long time intervals ($\tau \sim 400+$) the algorithm diverges. It doesn't necessarily mean that there is no solution to the problem, just that the algorithm isn't robust enough.

12 Matlab Algorithm #2

Therefore, I gave in and wrote another test programme that uses the built-in ode solver of matlab, namely 'matlabOdeSolver.m' (you can just run it to get the solution if there are variables for the constants in your work-space). Its results are mixed. It does always converge for any A_0 and other parameters, but for long intervals ($\tau \sim 150+$) it takes insanely long time to converge. I've let it run for about 15 minutes and got no result.

13 Discussion #2

Let's take a look at the best graph I managed to get, Graph 2. We see that the time that it takes to the AHL is extremely short, and LA and LA2 follow shortly afterwards. It takes much longer to TRLV to dissolve, and the increase of ccdb happens at the same time, more or less linearly. Obviously, this holds only for the parameters that have been used, but if we decide to use these parameters we can use the above mentioned analysis to know what is negligible and what isn't.

14 Future Research

No more work is needed. We are done. Go home.

Well, not quite yet. As you probably understood yourself, the results I got are very limited and insufficient. The convergence of the programme is unpredictable, and it's hard to conclude much about the effect of the various parameters. I by no means think that we can rely on the general solver that I wrote, and we have to continue with the analytical approach in order to at least simplify the programme before we run it on computer.

Anyway, here are some future tasks I can think of:

1. Simplify the problem with approximations, using the results from the simulation to know better how to divide to time intervals and which quantities are negligible.

2. Compute better estimation to the constants. I put some related links in the file 'Links.docx' in the root directory of the shared folder.

3. Analyse better the results of the simulation of Matlab. We need to understand better the effect of all the parameters on the result.

4. (Chen's suggestion) Write graphical user interface to the simulation in Matlab.